Theoretical Computer Science 408 (2008) 116–128

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

On the power of lookahead in on-line server routing problems

Luca Allulli [a,*], Giorgio Ausiello [a], Vincenzo Bonifaci [a,b], Luigi Laura [a]

[a] *Dipartimento di Informatica e Sistemistica, Sapienza Università di Roma, Via Ariosto 25, 00185 Roma, Italy*
[b] *Dipartimento di Ingegneria Elettrica, Università dell'Aquila, Monteluco di Roio, 67040 L'Aquila, Italy*

ARTICLE INFO

Keywords:
On-line algorithms
Competitive analysis
Lookahead
Traveling salesman problem

ABSTRACT

We study the usefulness of lookahead in on-line server routing problems: if an on-line algorithm is not only informed about the requests released so far, but also has a limited ability to foresee future requests, what is the improvement that can be achieved in terms of the competitive ratio? We consider several on-line server routing problems in this setting, such as the on-line traveling salesman and the on-line traveling repairman problem. We show that the influence of lookahead can change considerably depending on the particular objective function and metric space considered.

1. Introduction

In several practical circumstances we need to solve a problem without having, initially, complete knowledge of the problem instance, since the instance is gradually revealed over time. In such cases, the solution algorithm will operate on the basis of the known data and will progressively reorganize its behavior as new information is provided. Situations of this kind occur in computer systems and networks management, in financial decision making, in robotics etc. Problems that have to be solved without knowing the whole instance in advance are called *on-line problems* and the solution algorithms are called *on-line algorithms* [14,18]. In order to measure the quality of on-line algorithms, the notion of *competitive analysis* has been introduced, in which the value of the solution obtained by an on-line algorithm is compared to the value of the best solution that can be achieved by an optimum off-line algorithm having full knowledge of the problem instance ahead of time [32].

An important class of on-line problems that has received much attention in recent years, is the class of on-line vehicle routing problems. In this type of problems we imagine that a vehicle (also called *on-line server*) has to serve a sequence of requests which are released over time in a metric space, with the aim of minimizing a given objective function (e.g. the completion time). Examples of problems of this kind that have been extensively studied are the on-line traveling salesman problem (OL-TSP, [9]), the on-line traveling repairman problem (OL-TRP, [27]) and the on-line dial-a-ride problem [17].

A natural question that arises when dealing with on-line problems is whether providing an algorithm with limited clairvoyance, that is the capability to foresee future requests, may help in achieving a better competitive ratio. In this paper, we address the issue of determining how much clairvoyance helps an on-line server in the solution of vehicle routing problems, and we show positive and negative results, depending on the type of problem and the type of objective function. In particular, we show that, although for both the homing and the nomadic versions of OL-TSP [9] no time lookahead can guarantee a competitive ratio better than 2, in the nomadic case, classical on-line algorithms provided with time lookahead indeed outperform the correspondent versions without lookahead, both on the real plane and on the real line. In the case of the OL-TRP, instead, where the objective function is the net latency (or equivalently, the average waiting time of the requests)

* Corresponding author.
E-mail addresses: allulli@dis.uniroma1.it (L. Allulli), ausiello@dis.uniroma1.it (G. Ausiello), bonifaci@dis.uniroma1.it (V. Bonifaci), laura@dis.uniroma1.it (L. Laura).

0304-3975/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2008.08.003

we show, among others, a strong negative result that holds in any d-dimensional space, $d \geq 2$, stating that for such a space no constant competitive ratio can be achieved, whatever the size of the lookahead time window.

It is worth noting that a different, but related way in which an on-line algorithm can somehow emulate the capability to see future requests, is by behaving in a lazy manner, that is, not moving or moving at low speed and thus allowing the arrival of more information on the input instance before making decisions. In the paper we also discuss the relationships between clairvoyance and laziness in particular classes of on-line problems.

Such results concerning clairvoyance and laziness of on-line algorithms are particularly interesting for vehicle routing problems, and for other problems such as scheduling, in which the temporal aspect of the system cannot simply be captured by a sequence of discrete events, as is the case for other on-line problems, due to the fact that the time passing between two events cannot be neglected — usually because it influences the objective function. These problems are best modeled as *real-time* on-line problems [6].

The paper is organized as follows. In Section 2 the on-line version of the classical traveling salesman problem is introduced and competitiveness results for variants of this problem are reviewed. Besides, adversarial models that are motivated by the real-time context are also discussed. In Section 3 we introduce suitable notions of clairvoyance for the OL-TSP and we show positive and negative results for clairvoyant algorithms, while in Section 4 we show the limits to the power of clairvoyance in the case of OL-TRP. Finally in Section 5 we briefly discuss the relationship between clairvoyance and laziness in on-line algorithms for real-time on-line problems.

2. On-line server routing problems

In this section we present the basic notions and results related to the OL-TSP and its variations. We present problems in the classical framework provided by *competitive analysis*, and we also discuss alternative adversarial approaches. We conclude the section by defining *zealous algorithms*, and by discussing their performance in terms of competitive analysis.

The OL-TSP has been introduced by Ausiello et al. in [9]. In OL-TSP we are given a metric space $M = (X, d)$, where X is a set of points and d is a distance function on X, with a distinguished point $O \in X$, called the *origin*; and a set of requests $\sigma = \{\sigma_1, \ldots, \sigma_n\}$. Each request consists of a pair $\sigma_i = (x_i, t_i) \in X \times \mathbb{R}_0^+$, where x_i is the *position* of σ_i, and t_i is its *release time*. A *server* is located in the origin at time 0, and thereafter moves in the metric space, at most at unit speed, in order to *serve* all the requests, i.e. to visit each point x_i where a request is placed, not earlier than the release time t_i of the request. An additional constraint can be required that the server returns to the origin, after having served all the requests. The goal of the server is to find a feasible schedule that minimizes an *objective function*, which in some way measures the quality of the schedule.

As usual, the metric space M satisfies the following properties: (i) it is symmetric, i.e., for every pair of points x, y in M, $d(x, y) = d(y, x)$, where $d(x, y)$ denotes the distance from x to y; (ii) $d(x, x) = 0$ for every point x in M; (iii) it satisfies the triangle inequality, i.e., for any triple of points x, y and z in M it holds that $d(x, y) \leq d(x, z) + d(z, y)$. Furthermore the metric space M can be continuous, i.e., have the property that the shortest path from $x \in M$ to $y \in M$ is continuous, formed by points in M and has length $d(x, y)$. Examples of continuous metric spaces include the Euclidean plane, the real line or a line segment. A discrete metric space is represented by a metric graph in which all the edges have positive weights and requests are always located at the vertices.

Many objective functions have been proposed in the literature for the traveling salesman problem. Here we will mainly refer to the *completion time*, i.e. the time when the server completes its service, and the *latency*, i.e. the sum of the times each request has to wait to be served since time 0, namely $\sum_{i=1}^{n} \tau_i$, where τ_i is the time instant when request σ_i is served (see also [20] and [31]). Note that while the completion time is, so to say, a "selfish" measure, aimed at reducing the time spent by the server, latency can be considered an "altruistic" measure, aimed at reducing the overall waiting time of customers. If we consider the completion time, there are two distinct versions of the problem, depending on whether the server has to return to the origin at the end. These problems are known as the *Homing* on-line Traveling Salesman Problem (H-OL-TSP) and the *Nomadic*[1] on-line Traveling Salesman Problem (N-OL-TSP), respectively; we call on-line Traveling Repairman Problem (OL-TRP) the problem in which we want to minimize the latency [1].

We say that an on-line algorithm A is ρ-competitive ($\rho \in \mathbb{R}^+$) if, for any input instance σ, $\mathsf{A}(\sigma) \leq \rho \cdot \mathsf{OPT}(\sigma)$; we denote by $\mathsf{A}(\sigma)$ and $\mathsf{OPT}(\sigma)$ the cost, on input σ, of the solution found by A and of the optimal solution, respectively.

Table 1 contains an overview of the main competitiveness results concerning the problems defined above (the values are rounded to the second decimal digit). Considering the three problems, it clearly appears that the Homing version of TSP is in a sense the easiest one, because in all cases, competitiveness upper bounds matching the corresponding lower bounds have been established, while the Nomadic version still presents gaps between upper and lower bounds. Intuitively we can argue that this is due to the value of the information (implicitly exploited by the on-line server in H-OL-TSP) that the adversary has to return to the origin at the end of its tour, information that is lacking in N-OL-TSP. More interesting appear the large gaps still existing in the case of the latency problem, both for general metric spaces and for particular metric spaces, such as the real line; these gaps resist as the major open problems in this domain.

[1] Also known as the Wandering Traveling Salesman Problem [24].

Table 1
Known competitiveness results

H-Ol-Tsp			
Metric space	Lower bound	Upper bound	Ref.
General	2	2	[9]
Real line	1.64	1.64	[30]
N-Ol-Tsp			
Metric space	Lower bound	Upper bound	Ref.
General	2.03	2.42	[30]
Real line	2.03	2.06	[30]
Ol-Trp			
Metric space	Lower bound	Upper bound	Ref.
General	2.41	5.83	[17,27]
Real line	2.41	5.83	[17,27]

Minimizing the net latency. A problem strictly related to the Ol-Trp is the NL-Ol-Trp, in which the objective function to minimize is the *net latency*, i.e. the sum of the times each request has to wait to be served since its release time, namely $\sum_{i=1}^{n}(\tau_i - t_i)$. Note that, if we define $T = \sum_{i=1}^{n} t_i$ to be the sum of all the release times, that is a constant term, it is easy to see that the objective function can be rewritten as $\sum_{i=1}^{n}(\tau_i - t_i) = \left(\sum_{i=1}^{n} \tau_i\right) - T$ that is the latency minus T, i.e. the latency minus a constant term; therefore minimizing latency or net latency should be exactly the same. However, constant terms alter the competitive ratio, and it is not hard to see that there cannot be a competitive algorithm for the NL-Ol-Trp.

Proposition 1. *There is no competitive algorithm for the* NL-Ol-Trp.

Proof. Consider the real line as the metric space. Assume wlog, that at time 1 the on-line server is in the positive half of the line: a request is released in position -1, and the adversary serves it immediately. There are no other requests, and the net latency of the adversary is 0, while the on-line server pays a positive cost. □

We will further discuss the NL-Ol-Trp in Section 4.

Related problems. The traveling salesman problem can be seen as a special case of a broader family of vehicle routing problems known as *dial-a-ride*: here a server, in a metric space, is presented a sequence of *rides*; each ride is a triple $\sigma_i = (t_i, s_i, d_i)$, where t_i is the time at which the ride σ_i is released, and s_i and d_i are, respectively, the *source* and the *destination* of the ride. Every ride has to be executed (served) by the server, that is, the server has to visit the source, start the ride, and end it at the destination. The capacity of the the server is an upper bound on the number of rides the server can execute simultaneously. In the literature unit capacity, constant capacity $c \geq 2$, and infinite capacity for the server are usually considered. This family of problems can be used to model, for example, a taxi service (unit capacity), an elevator scheduling and delivery service (constant capacity) or a postal service (infinite capacity). Ascheuer, Krumke and Rambau [5] and, independently, Feuerstein and Stougie [17] started the study of on-line dial-a-ride problems, and up to date results can be found in [16,30].

Another generalization of the Ol-Tsp is the well known *Quota* Tsp problem (a generalization of the k-Tsp [19]): here the goal of the traveling salesman is to reach a given *quota* of sales, while minimizing the traveling time. The on-line Quota Tsp has been addressed in [8], where best possible bounds and algorithms for several metric spaces are presented.

Another direction in which the Ol-Tsp can be generalized, is by dropping the constraint that the underlying space is symmetric (while maintaining the triangle inequality). This way one obtains the on-line Asymmetric Tsp. This problem has been studied in [7] both in the homing and nomadic version: for the former, the authors provide a best possible competitive algorithm; for the latter, they show that in general no on-line competitive algorithm is possible; indeed, the competitive ratio has to be a function of the amount of asymmetry of the space, i.e. of the smallest K such that $d(x, y) \leq Kd(y, x)$ for all locations x, y.

Alternative adversarial models. It is well known that competitive analysis has been criticized for being too pessimistic, since it is often possible to build up pathological input instances, that only an off-line server can serve effectively, thanks to its clairvoyance. Competitive analysis can be seen as a game between the on-line algorithm and an off-line adversary: the latter builds up an input instance that is difficult for the on-line algorithm, while serving it effectively. Using such a metaphor, the off-line adversary is often too powerful with respect to the on-line algorithm. In order to limit, in some way, the power of the off-line adversary, restricted types of adversary have been proposed that are not allowed to behave in an excessively unfair way with respect to the on-line algorithm. Here we mention only the ones that are specific in the context of on-line real-time problems.

Blom et al. [10] introduce the *fair* adversary, that is restricted to keep its server within the convex hull of the requests released so far. In this way sequences like the one we present in the proof of Theorem 4 are no longer allowed: it is not

Table 2
The competitiveness of *zealous* algorithms

H-OL-TSP			
Metric space	Lower bound	Upper bound	Ref.
General	2	2	[9]
Real line	1.75	1.75	[9,10]
N-OL-TSP			
Metric space	Lower bound	Upper bound	Ref.
General	2.05	2.5	[9,30]
Real line	2.05	2.33	[9,30]
OL-TRP			
Metric space	Lower bound	Upper bound	Ref.
General	3	open	[29]
Real line	3	open	[29]

possible for the adversary to move its server "without an evident reason" from the perspective of the on-line player. The authors show that against the fair adversary, the on-line server achieves better competitive ratios.

Krumke et al. [28] propose the *non-abusive* adversary for the on-line TSP, where the objective is to minimize the maximum flow time, i.e. $\max_i(\tau_i - t_i)$; note that for this problem there are no competitive algorithms against general (or fair) adversaries. A *non-abusive adversary* may only move in a direction, if there are yet unserved requests on this side. Krumke et al. present an algorithm that is competitive against the non-abusive adversary.

An alternative technique for overcoming the excessive power of the adversary, frequently used in on-line optimization, is the so called *resource augmentation*: instead of limiting the power of the adversary, the idea is to increase the resources of the on-line algorithm, such as speed or number of servers. Resource augmentation has been used for on-line problems such as scheduling since the early work of Graham [21]. See [12,13] for resource augmentation results for on-line vehicle routing problems.

A completely different approach to avoid pathological worst case input sequences, consists in assuming a bound on the rate with which requests can be injected into the system. This approach has been pursued first in [22]. Later, in [11], a similar approach has been pursued in the larger context of *adversarial queueing theory* [15].

Zealous algorithms. A peculiarity of real-time on-line problems, like the ones we discuss in this paper, is that a server is allowed to decide whether to serve a request or not, and it can even wait idle. At a first glance, it may sound unusual that an algorithm should decide to wait instead of serving pending requests; but consider the following case: the server is in the origin, and the only request released so far is far away from its current position; therefore it seems not a bad idea to "wait a little", or alternatively to move "slowly" towards it, to see if other requests show up in order to serve all of them together. Here the real-time aspect of the problem combines with the fact that moving a server could damage the quality of the overall service; this might not happen if we consider other real-time problems like scheduling, if we allow jobs to be interrupted (even if we might start them again from scratch later). Now, if we concentrate on vehicle routing problems, the benefits of waiting could depend on the objective function; intuitively, if we want to minimize latency it could be more "dangerous" to move the server towards an isolated request far away, while, if completion time is the objective function, serving a distant request might be less insecure.

How can we measure, in a real-time problem, the importance of waiting, or, more precisely, the importance of the capability to wait? To do so, we recall from the work of Blom et al. [10] the notion of *zealous algorithm* for on-line routing problems; informally, a zealous algorithm is not allowed to wait.[2]

Definition 2 (*Zealous Algorithm*). When there are unserved requests, the direction of a server operated by a zealous algorithm (a zealous server) changes only if a new request becomes known, or if the server is either in the origin or it has just served a request. A zealous server is allowed to move only at maximum (i.e., unit) speed.

Zealous algorithms are a natural and well-defined class of algorithms, and they are usually easy to analyze, because of their restricted behavior. Such analyses are useful to measure the importance of waiting by studying how much, for a given problem, the algorithms that are not allowed to wait are penalized. In Table 2, we summarize the best bounds known for zealous algorithms. By comparing these results with those in Table 1, one may observe that non-zealous algorithms perform better in all but the first case. This corresponds to the intuition that, in on-line server routing problems, waiting usually helps (see also [30]).

[2] Originally, in [10], the authors used the term *diligent* instead of zealous.

3. The on-line TSP with lookahead

The standard concept of lookahead, as originally defined for problems in the step-by-step model of on-line computation (see for example [2,3,25]) must be reconsidered in order to obtain meaningful results in our real-time model. In the step-by-step setting *request lookahead* is typically used: the on-line algorithm can see, at any time, the next k requests that will be released in the future, for some $k \in \mathbb{N}$. If we export this definition to real-time problems, we obtain a concept of lookahead that is unrealistic and scarcely meaningful. Unrealistic is the assumption that a real-world application would be able to see the next k requests independently of when they will be released. Scarcely useful because it is unlikely that the quite bizarre additional information provided by request lookahead yields meaningful performance improvement to on-line algorithms [4].

A different, more natural definition of lookahead for real-time problems is time lookahead, that we introduce in the following.

Definition 3 (*Time Lookahead*). An on-line algorithm A for a real-time problem has *time lookahead* $\Delta \in \mathbb{R}^+$ if, at any time $t \geq 0$, A has received in input all the requests with release time at most $t + \Delta$.

An algorithm with time lookahead Δ can see what happens in a time window of length Δ in the future. Observe that, in order to be a meaningful value, Δ should be related to some characteristic quantity of the problem or of the instance. Our results depend on the ratio between Δ and the time that a server employs to traverse the entire metric space, i.e. the diameter D of the metric space (under the usual assumption that the server moves at unit speed). Obviously this kind of analysis makes sense if the metric space is bounded. Jaillet and Wagner [23] give a very similar definition of lookahead, but they essentially compare Δ with the optimal cost of each input instance. We will discuss their approach further on in Section 5.

3.1. General metric spaces

We now study the influence of time lookahead in the OL-TSP. We begin with a lower bound, which holds for both the homing and the nomadic variants of the problem. We prove that no on-line algorithm for the OL-TSP can be better than 2-competitive in the general metric space.

While the result is the same for both the variants of the OL-TSP, its impact is very different. In the homing version, an optimal 2-competitive algorithm without lookahead exists, as shown by Ausiello et al. [9]: thus lookahead is useless in this case. Instead, our lower bound leaves room for improving the nomadic case, because it is smaller than the current lower bound of about 2.03 for algorithms with no lookahead [30] (which is not matched by any algorithm, currently). We will later show that lookahead is indeed useful in the nomadic case, and that the lower bound of 2 is matched for a sufficiently large value of Δ.

Before going into the proof, let us remark the bad news: the lower bound of 2 holds for any value of Δ. It is a bit surprising that large amounts of lookahead do not help to further improve the competitive ratio of the problem.

The lower bound of 2 for the H-OL-TSP without lookahead has been proved by Ausiello et al. [9] first, and later, with a different proof, by Lipmann [30]. Our proof is inspired by the second approach.

Theorem 4. *No deterministic algorithm for the* H-OL-TSP *or the* N-OL-TSP *can be better than* 2*-competitive, independently of the amount of time lookahead.*

Proof. Consider a star graph $G = (V, E)$ with $N+1$ nodes: a central node v_0 and N peripheral nodes $v_1, \ldots, v_N$ (see Fig. 1). Each peripheral node v_i is connected to the central node by an edge $e_i = \{v_0, v_i\}$ of length $1/2$. Let A be any algorithm for the H-OL-TSP or the N-OL-TSP on G with time lookahead Δ.

At time Δ, N requests are presented, one in each peripheral node. Let $t_{stop} = \Delta + N - 1$. For any time $t \leq t_{stop}$, if A serves one request in vertex v_i at time t, then a new request is presented in the same vertex v_i at time $t + \Delta$; A can see it immediately, according to its lookahead. Thus, at any time $t \leq t_{stop}$, A is aware of exactly N requests that either have been released but not served or will be released in the future. In particular, at time t_{stop}, A must still serve N requests, and cannot finish before time $t_{stop} + N - 1 = \Delta + 2N - 2$.

On the other hand, an off-line adversary can complete its service not later than time $2\Delta + N$. In fact, it can serve requests in the following order: first the requests in those vertices that are never touched by A before time t_{stop}, if any; then, all the other requests, visiting peripheral vertices once and in the same order A visits them for the last time. This way, giving to the off-line adversary a delay of at least Δ over A, it can serve the newly presented requests in every peripheral vertex along with the old ones. Thus the adversary finishes not later than time $(1/2 + 2\Delta) + (N - 1) + 1/2$: the first term is a time sufficient to reach the first request and to gain a delay of Δ over A; the second term is a time sufficient to serve all the requests, and the last term is a time sufficient to return home, if the problem is the H-OL-TSP.

The lower bound on the competitive ratio

$$\frac{\mathsf{A}(\sigma)}{\mathsf{OPT}(\sigma)} \geq \frac{2N + \Delta - 2}{N + 2\Delta}$$

can be made arbitrarily close to 2 by choosing a sufficiently large N. □

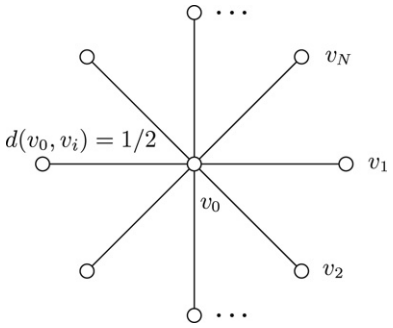

Fig. 1. The star graph G.

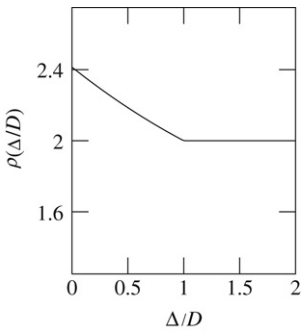

Fig. 2. The competitive ratio of ReturnHome as a function of Δ/D.

We now focus on the N-Ol-Tsp case, for which we show that lookahead is indeed helpful. We give an algorithm that matches the lower bound of 2 when $\Delta = D$, i.e. when there is enough lookahead so that the server can traverse the entire metric space when it foresees a new request, and reach the request before its release time. We remark that the current best algorithm for the N-Ol-Tsp without lookahead is the $(1 + \sqrt{2})$-competitive ReturnHome by Lipmann [30]. The competitive ratio of our algorithm, which is indeed a natural extension of ReturnHome, depends on its amount Δ of lookahead: it is $1 + \sqrt{2}$ when $\Delta = 0$, it continuously and monotonically decreases when $\Delta \in [0, D]$ and it remains 2 for $\Delta \geq D$ (Fig. 2).

Like ReturnHome, our algorithm (Algorithm 1) depends on a parameter α. It is not a zealous algorithm: in order to remain sufficiently close to the origin, so that it can quickly come back "home" when a new request is foreseen, it will stay at any time t within a ball of radius αt centered in the origin. The optimal value for the parameter α depends on the ratio $\delta = \Delta/D$.

Algorithm 1 $\mathsf{ReturnHome}_\alpha$ with time lookahead Δ

At every time $t \in \mathbb{R}^+$, algorithm $\mathsf{ReturnHome}_\alpha$ (RH_α) is either idle or it is following a tour T. RH_α is a parametric algorithm, with parameter $\alpha \in (0, 1]$. RH_α maintains the invariant that, at any time t, its server stays within a distance of αt from the origin (*ball-constraint*). This is achieved by always moving at the highest possible speed that does not violate the constraint (see also Example 5).

Initially, RH_α is idle. Independently of its current state, as soon as RH_α foresees a new request according to its lookahead, it immediately returns to the origin, and waits for the new request to be actually released. Then, it begins to follow the minimum-length tour T over all the released, but not yet served requests.

Example 5. Assume that $\alpha = 1/2$ and $\Delta = 0$. Consider an instance on the Euclidean line $\mathbb{R}$, with a request released at time 1 in point $+2$. $\mathsf{ReturnHome}_\alpha$ waits idly in the origin until time 1, then it starts moving at full speed towards the request, up to time 2. At time 2 the server is at distance $1 = \alpha \cdot 2$ from the origin and the ball-constraint becomes active. The server proceeds at speed 1/2 up to point $+2$, which is thus reached only at time 4. If at this time a new request is released in the origin, the server moves at unit speed towards the origin, unhampered by the ball-constraint.

Theorem 6. *For every $\delta \geq 0$, there is $\alpha \in (0, 1]$ such that RH_α is an algorithm with lookahead δD which is $\rho(\delta)$-competitive for* N-Ol-Tsp *on any metric space with diameter D, where*

$$\rho(\delta) = \max\left\{2,\ 1 + \frac{1}{2}\left(\sqrt{\delta^2 + 8} - \delta\right)\right\}.$$

Proof. We distinguish two cases, depending on whether the server has to reduce its speed during the last tour, i.e. the tour scheduled after the last request is released.

Case 1. The server reduces its speed during the last tour.
Let $\sigma_r = (x_r, t_r)$ be the last request that forces RH_α to regulate its speed. Since RH_α regulates its speed only when it is necessary, σ_r is served at time $\frac{x_r}{\alpha}$. Thereafter, RH_α completes the optimum tour at full speed: let $|T_r|$ be the length of the remaining part of the tour. We have that $\mathsf{RH}_\alpha(\sigma) = \frac{x_r}{\alpha} + |T_r| \leq \frac{1}{\alpha}(x_r + |T_r|)$. On the other hand, the off-line adversary pays at least $x_r + |T_r|$ to get to x_r and serve the remaining requests. Therefore: $\frac{\mathsf{RH}_\alpha(\sigma)}{\mathsf{OPT}(\sigma)} \leq \frac{1}{\alpha}$.

Case 2. The server does not reduce its speed during the last tour.
Assume that the last request is released at time $t + \delta D$; it is foreseen by RH_α at time t, thanks to its lookahead. RH_α will immediately come back to the origin, and will start following the optimal tour T at the first time $t_0 \geq t + \delta D$ when the server is in the origin. The completion time of RH_α is thus $\mathsf{RH}_\alpha(\sigma) = t_0 + |T|$. Now we give two lower bounds on the optimal cost. Since the off-line adversary must visit all the requests, it pays at least $|T|$. Since it must serve the last request not earlier than its release time, it pays at least $t + \delta D$. Hence, the competitive ratio of RH_α can be bounded by:

$$\frac{\mathsf{RH}_\alpha(\sigma)}{\mathsf{OPT}(\sigma)} = \frac{t_0 + |T|}{\mathsf{OPT}(\sigma)} \leq \frac{t_0}{t + \delta D} + \frac{|T|}{|T|} = \frac{t_0}{t + \delta D} + 1.$$

In the following, we give an upper bound on $\frac{t_0}{t+\delta D}$. We distinguish two subcases, depending on t. Intuitively, if t is small, i.e. not much time has elapsed since time 0, the server is near the origin and can quickly return home; otherwise, t_0 is comparable to $t + \delta$, because the metric space is bounded and RH_α cannot be too far away from the origin.

- If $\alpha t \leq D$, we use the fact that

  $$t_0 \leq \max\{t + \alpha t, t + \delta D\}$$

  thanks to the ball constraint. Thus $\frac{t_0}{t+\delta D} \leq \max\{\frac{t+\alpha t}{t+\delta D}, 1\}$; the latter quantity is monotonically increasing in t, and reaches its maximum, $\max\{1 + \alpha\frac{1-\delta}{1+\alpha\delta}, 1\}$, when $\alpha t = D$.
- If $\alpha t > D$, we use the fact that

  $$t_0 \leq \max\{t + D, t + \delta D\},$$

  because the metric space has diameter D. Thus $\frac{t_0}{t+\delta D} \leq \max\{\frac{t+D}{t+\delta D}, 1\}$; the latter quantity is monotonically decreasing in t, and, as before, the maximum of the expression is $\max\{1 + \alpha\frac{1-\delta}{1+\alpha\delta}, 1\}$.

  From Case 1 and Case 2 we infer that

  $$\frac{\mathsf{RH}_\alpha(\sigma)}{\mathsf{OPT}(\sigma)} \leq \max\left\{\frac{1}{\alpha},\ 2 + \alpha\frac{1-\delta}{1+\alpha\delta}, 2\right\}.$$

  For any value of $\delta \geq 0$, this expression is minimized by choosing $\alpha = \frac{\sqrt{\delta^2+8}+\delta-2}{2(\delta+1)}$. □

Notice that whenever the lookahead is at least as large as the diameter of the space (i.e., $\delta \geq 1$) the Theorem yields a 2-competitive algorithm, thus matching the lower bound of Theorem 4.

3.2. The line segment

As we said earlier, the general lower bound of 2 for large amounts of lookahead is rather disappointing. Fortunately there are specific metric spaces where lookahead plays a more natural role: the larger the amount of lookahead, the better the competitive ratio of the algorithms. One such metric space is the one-dimensional interval, or line segment.

We now present a simple algorithm for the line segment with a competitive ratio that tends to 1 as $\delta = \Delta/D$ increases. Our algorithm is parametric with respect to an objective function: it contains an optimization step where optimization is performed with respect to the chosen function. This allows us to tune the algorithm for both the homing and the nomadic version of the OL-TSP, and, as we will see in the next section, for the OL-TRP.

Algorithm 2 $\mathsf{OptimizeEarlierRequestsOnly}_f$

At time 0, algorithm $\mathsf{OptimizeEarlierRequestsOnly}_f$ (henceforth simply OERO) foresees all the requests that will be released up to time Δ. It computes the optimal schedule over these requests, with respect to the objective function f, and begins to follow it. After time Δ, if new requests are released, then OERO switches to another mode, even if it has not completed the scheduled tour: it continuously sweeps the line segment from one extreme to the other at full speed, serving all the requests it encounters.

Theorem 7. *If f is the completion time in the nomadic case,* $\mathsf{OptimizeEarlierRequestsOnly}_f$ *with time lookahead $\Delta = \delta D$ is a $(1 + 2/\delta)$-competitive algorithm for the* N-OL-TSP *defined on an interval of length D.*

Proof. Let σ be the input instance. If no requests are released after time Δ, OERO is clearly 1-competitive. Otherwise, we have that $\mathsf{OPT}(\sigma) \geq \delta D$. Let $\sigma^* = (x^*, t^*)$ be the last request served by OERO. Since after time Δ an active request waits at most $2D$ time units before being served, we have that $\mathsf{OERO}(\sigma) \leq t^* + 2D$. Obviously $\mathsf{OPT}(\sigma) \geq t^*$. Then we obtain $\mathsf{OERO}(\sigma) \leq \mathsf{OPT}(\sigma) + (2/\delta)\mathsf{OPT}(\sigma) = (1 + 2/\delta)\mathsf{OPT}(\sigma)$. □

For the H-OL-TSP one further specification is needed. As soon as all the active requests have been served, OERO comes back to the origin; if afterwards other requests appear, OERO resumes sweeping the interval. The result is the following.

Theorem 8. *If f is the completion time in the homing case,* $\mathsf{OptimizeEarlierRequestsOnly}_f$ *with time lookahead $\Delta = \delta D$ is a $(1 + 2/\delta)$-competitive algorithm for the* H-OL-TSP *defined on an interval of length D.*

Proof. Let σ be the input instance. If no requests are released after time Δ, OERO is clearly 1-competitive. Otherwise, we have that $\mathsf{OPT}(\sigma) \geq \delta D$. Let $\sigma^* = (x^*, t^*)$ be the last request served by OERO. Since after time Δ an active request waits at most $2D$ time units before being served, we have that $\mathsf{OERO}(\sigma) \leq t^* + 2D + d(O, x^*)$. Obviously $\mathsf{OPT}(\sigma) \geq t^* + d(O, x^*)$. Then we obtain $\mathsf{OERO}(\sigma) \leq \mathsf{OPT}(\sigma) + (2/\delta)\mathsf{OPT}(\sigma) = (1 + 2/\delta)\mathsf{OPT}(\sigma)$. □

4. The on-line TRP with lookahead

4.1. General metric spaces

As we discussed earlier (Proposition 1), no on-line competitive algorithm for the net latency objective function exists. The reason for such a negative result is basically the excessive power of the off-line adversary, which essentially can issue a request wherever it wants and serve it at no cost, while the on-line algorithm must traverse the metric space to reach the request. In order to be "fair", it is very natural to request the adversary to disclose requests in advance.

Lookahead makes sense, even from a practical point of view. For most real world applications modeled by a vehicle routing problem where net latency would be a meaningful objective function, it is reasonable to assume some form of lookahead: customers asking for good service should notify their requests in advance.

Consequently, we wonder whether a sufficiently large amount of lookahead allows an algorithm to be competitive for this important objective function. Unfortunately the answer is negative, at least in any metric space in which we can embed a bidimensional ball: in the next theorem we show that no competitive on-line algorithm exists for the NL-OL-TRP in this case, independently of the amount of lookahead.

Theorem 9. *Let Ω be a open set of $\mathbb{R}^n$, $n \geq 2$; let* A *be an on-line algorithm for the* NL-OL-TRP *on Ω with time lookahead Δ. Then, for all $\Delta \in \mathbb{R}^+$,* A *is not competitive for the* NL-OL-TRP *in Ω.*

Proof. Let us introduce some notation. We will refer to the on-line algorithm as A, to the adversary as B (with slight abuse we use the terms "algorithm" and "server" as synonyms). For any server $Y \in \{\mathsf{A}, \mathsf{B}\}$, we will denote by $p_Y(t)$ the position of Y at time t.

Here is an overview of the proof. We will construct a set G of $2N$ points such that, in order to visit any subset of N points, a minimum time of Δ is needed. The adversary will release some initial requests: at least one request in each point of G. Furthermore, the adversary will select a subset $G^- \subset G$ containing N points, and will force A to serve the requests in G^- first: otherwise A will not be competitive. While A serves the starting requests in G^-, B serves all the other starting requests; afterwards, B begins to follow A with a delay of Δ. In the meanwhile, new requests are generated. If A serves some requests at time t, then B releases a new request in the same point at time $t + \Delta$. B is able to serve all the new requests on the fly, thanks to its delay of Δ with respect to A; on the other hand, A is continuously late, in the sense that in every time interval of length 2Δ, there are active requests that cannot be served by A as soon as they are released.

Let us construct the set G. Without loss of generality, we assume that $\Omega \subseteq \mathbb{R}^2$. The points in G will be the elements of a $k \times k$ square grid (where $k^2 = 2N$, and k is an even number) with side of length a, contained inside a ball $B_O\left(\frac{\sqrt{2}}{2}a\right) \subseteq \Omega$ with diameter $\sqrt{2}a \leq \Delta$ (see Fig. 3). The minimum distance between any two points of G is thus $d = a/k$. In order to have $\Delta \leq d \cdot N = (a/k)(k^2/2) = ak/2$, it suffices to take $k \geq \left\lceil \frac{2\Delta}{a} \right\rceil$.

We can assume the coordinates of O being $(0, 0)$. In order to refer to the points of G, for each $i \in \{-k/2, \ldots, -2, -1, +1, \ldots, +k/2\}$ we denote by P_{ij} the point of coordinates $(f(i), f(j))$ (see Fig. 3), where

$$f(i) = \begin{cases} i\frac{a}{k} - \frac{a}{2k}, & 1 \leq i \leq \frac{k}{2}; \\ i\frac{a}{k} + \frac{a}{2k}, & -\frac{k}{2} \leq i \leq -1. \end{cases}$$

We partition G into two subsets of cardinality N each: the set $G^+ = \{P_{ij} \in G | j > 0\}$ containing the points of G located "above" the origin, and the set $G^- = G - G^+$ containing points located "below" the origin.

The *Starting requests* are the first requests released by B. The sequence $\sigma^{(0)}$ of the starting requests has the following properties: (i) each request $(t, x) \in \sigma^{(0)}$ has a release time $t \in [\Delta, 2\Delta)$, and (ii) for every point $P_{ij} \in G$ there exists a request $(t, x) \in \sigma^{(0)}$ such that $x = P_{ij}$.

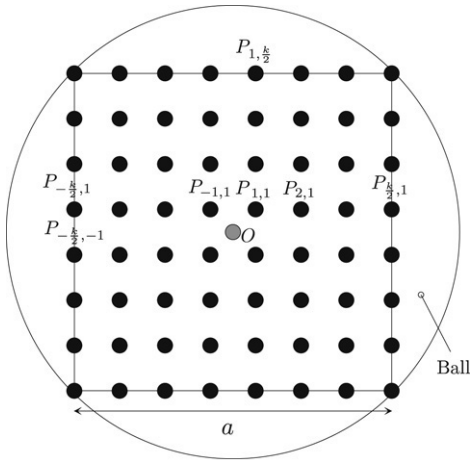

Fig. 3. The grid G.

Let $L(t) = \{(t_k, x_k) \in \sigma \mid t_k - \Delta \leq t < \tau_k\}$ be the set of requests which either have been released but not yet served by A at time t, or that will be released before time $t + \Delta$. $L(t)$ contains all the unserved requests whose existence is known by A at time t, according to its lookahead. Moreover, if $S \subseteq \sigma$, let $G_S = \{P_{ij} \in G \mid \exists (t_k, x_k) \in S \text{ s.t. } x_k = P_{ij}\}$ be the set of points where at least one request of S is located. Note that $|G_{L(\Delta)}| = 2N$, since at time $t = \Delta$ in each point of G there is at least one (unserved) request that will be released not later than time 2Δ.

In addition to the requests of $\sigma^{(0)}$, B releases exactly one new request at each time $t \leq t_{stop}$ when A serves some requests located in $P_{ij} \in G$. The new request is $(t + \Delta, P_{ij})$. t_{stop} is a time that will be determined later. It is clear that $|G_{L(t)}| = 2N \quad \forall t \in [\Delta, t_{stop}]$. In fact, every time a request is served in P_{ij}, A becomes aware of a new request in the same point P_{ij}: this implies that $G_{L(t)} = G \quad \forall t \in [\Delta, t_{stop}]$.

The release times of starting requests are chosen in such a way that A *may* serve before time 2Δ all the starting requests that are located in G^-; furthermore, the very first requests served by A may be those located in $P_{-\frac{k}{2},-1}$. We will refer to this behavior as A's *expected behavior*. It is easy to show that B can choose $\sigma^{(0)}$ such that, if A does not have the expected behavior, then A is not competitive; this claim will be proved later. Hence, we suppose that A exhibits the expected behavior.

We shall give a lower bound on the cost incurred by A. We divide the "interesting portion" of the time axis into m time intervals $\mathcal{I}_2, \mathcal{I}_4, \ldots, \mathcal{I}_{2m}$ of width 2Δ each: for every i, we let $\mathcal{I}_i = [T_i, T_{i+2})$, where $T_i = i \cdot \Delta$; the last interval ends at time $T_{2m+2} = t_{stop}$. Now, consider the interval $\mathcal{I}_h$. At time T_h, at the beginning of $\mathcal{I}_h$, $G_{L(T_h)} = G$. In the first half of $\mathcal{I}_h$, in Δ time units, not more than N of the $2N$ points of G can be visited by A: so, at time T_{h+1} the remaining points (at least N points) contain a request that (i) has already been released, since it was in $L(T_h)$, and (ii) has not been served by A yet. Among these points, the i-th point visited by A cannot be reached earlier than time $T_{h+1} + (i-1)d$; A pays at least $(i-1)d$ for the corresponding request(s). Notice that this cost is totally paid inside interval $\mathcal{I}_h$; if a request $\sigma_j \in L(T_h)$ is still active at the beginning of the following interval $\mathcal{I}_{h+2}$, then σ_j will *also* contribute to the cost paid by A for the requests in $L(T_{h+2})$. Thus, in $\mathcal{I}_h$, A pays at least $\sum_{i=1}^{N}(i-1)d = \frac{N(N-1)}{2}\frac{\Delta}{N} = \frac{(N-1)\Delta}{2}$ and, since there are m intervals:

$$\mathsf{A}(\sigma) \geq m\frac{(N-1)\Delta}{2}.$$

We now describe the behavior of B (see Fig. 4), under the assumption that A has the expected behavior.

(1) During the interval $[0, \Delta)$, B moves its server to the first request to serve.
(2) During the interval $[\Delta, 2\Delta)$, B serves all requests located in G^+, scanning horizontally the grid row by row; B terminates these visits in point $P_{-\frac{k}{2},1}$, and afterwards moves its server to the immediately underlying point $P_{-\frac{k}{2},-1}$. B is able to reach $P_{-\frac{k}{2},-1}$ at time 2Δ.
(3) Recall that the first requests served by A, say at time $t_{start} > \Delta$, are located in $P_{-\frac{k}{2},-1}$. From time $t_{start} + \Delta$ on, B follows A with a delay of Δ, that is: $p_{\mathsf{B}}(t) = p_{\mathsf{A}}(t - \Delta) \quad \forall t \geq t_{start} + \Delta$. B serves all the active requests it encounters.

At time 2Δ all the starting requests in G^+ have been served by B, and at time 3Δ the starting requests in G^- have been served too (because A serves the starting requests in G^- before time 2Δ). Thus, at time 3Δ, B has served all the starting requests, paying some cost B_0 for them.

Any other request $\sigma_j = (t_j, x_j) \in \sigma - \sigma^{(0)}$ is served by B at no cost, because $x_j = p_{\mathsf{A}}(t_j - \Delta) = p_{\mathsf{B}}(t_j)$. We have that

$$\mathsf{B}(\sigma) = B_0,$$

and B_0 does not depend on t_{stop}. For any $\rho \in \mathbb{R}^+$, we can enforce $\mathsf{A}(\sigma)/\mathsf{B}(\sigma) > \rho$ by taking a large enough value for m.

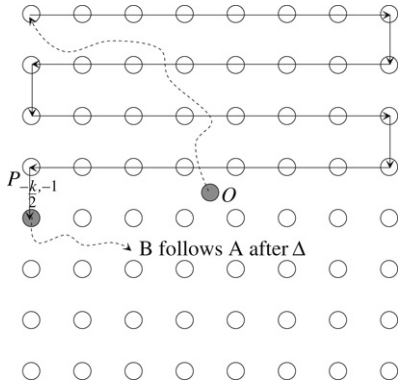

Fig. 4. B's server path.

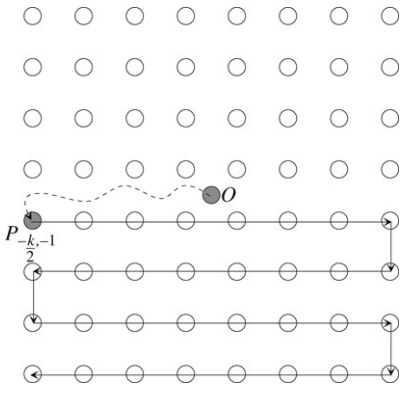

Fig. 5. A_v's path.

It remains to be shown that, if A does not have the expected behavior, then A is not competitive. In this situation, B selects $t_{stop} = 2\Delta$; notice this does not influence A's behavior before time 2Δ. Let A_v be an imaginary server which exhibits the expected behavior. In particular, at time Δ, A_v is in $P_{-\frac{k}{2},-1}$; between Δ and $2\Delta - d$, A_v serves all the initial requests located in G^-, scanning the grid horizontally one row after the other (see Fig. 5).

Now we shall give the complete definition of $\sigma^{(0)}$. In each point of G^+ exactly one request is released at time Δ; in each point $P_{ij} \in G^-$ exactly M requests are released at the first time $t \geq \Delta$ when $p_{A_v}(t) = P_{ij}$. Since A does not have the expected behavior, there exists at least one point $P_{ij} \in G^-$ where A serves M starting requests with a minimum delay of d. This means that

$$A(\sigma) \geq Md = M\frac{\Delta}{N}.$$

On the other hand, B behaves in a completely different manner than before: during time interval $[0,\ 2\Delta]$, B follows A_v, serving all the requests in G^- as soon as they are presented, at no cost; during the interval $(2\Delta,\ 3\Delta)$ it is idle; during the interval $[3\Delta,\ 5\Delta]$ it scans the whole grid once again, serving all the yet unserved requests (which are at most $2N$). Thus

$$B(\sigma) \leq 2N \cdot 4\Delta,$$

and, for any $\rho \in \mathbb{R}^+$, the ratio $A(\sigma)/B(\sigma) > \rho$ can be made arbitrarily large by taking a large enough value for M. □

4.2. The Line Segment

The technique we used to prove the last theorem is based on the bi-dimensional density of $\mathbb{R}^2$, which makes it possible to force the on-line algorithm to take an arbitrarily long tour in order to serve all the requests, independently from the diameter of the metric space. In other kinds of metric spaces, such as uni-dimensional or discrete spaces (the bounded real line is one notable example), the same technique cannot be used. Anyway, we now show that no algorithm can be competitive in any metric space, even with time lookahead, if its lookahead is less than two times the diameter of the metric space. This is still quite a large amount of lookahead for many real world applications.

Theorem 10. *Let $M = (X,\ dist)$ be any metric space with diameter D, and* A *any algorithm for the* NL-OL-TRP *on M with time lookahead Δ. If $\Delta < 2D$, then* A *is not competitive.*

Proof. We denote by $p_Y(t)$ the position of the server moved by algorithm Y at time t. In this proof we assume, without loss of generality, that $\Delta > D$.

Consider two points $P,\ Q \in X$ such that $d := dist(P,\ Q) > \Delta/2$. Notice that $d \leq D < \Delta < 2d$. The off-line adversary, B, releases all the requests in points P and Q. At time Δ it releases two requests: $\sigma_1 = (\Delta,\ P)$ and $\sigma_2 = (\Delta,\ Q)$. We refer to these requests as the *starting requests*.

Let $t_{stop} > \Delta$ be a time that will be determined later. Up to time t_{stop}, if at time $t \leq t_{stop}$ the on-line algorithm A serves a request in P (or Q), then the adversary releases a new request in P (resp. Q) at time $t + \Delta$. No other requests are released.

It is clear that, for every time $t \leq t_{stop}$, there are exactly two requests that either have been already released but not yet served by A, or that will be released not later than time $t + \Delta$. One request is located in P, and the other one in Q. In other words, according to its lookahead of Δ, A can see two unserved requests, one in P and the other one in Q. In fact, this is true for time 0, when A can see the starting requests; and every time A serves a request, a new request appears with a time distance of Δ.

Consider the requests in the order they are served by A, and let τ_i be the time when A serves the i-th such request. We divide the "interesting portion" of the time axis into intervals $\mathcal{I}_2,\ \mathcal{I}_4,\ \ldots,\ \mathcal{I}_{2m}$, where $\mathcal{I}_i = [\tau_i,\ \tau_{i+2}]$, and we select $t_{stop} = \tau_{2m+2}$ (so that $\mathcal{I}_{2m}$ ends at time t_{stop}). We shall give a lower bound on the cost incurred by A during the generic interval $\mathcal{I}_i$; in other words, we want to calculate how much time the active requests have to wait inside $\mathcal{I}_i$.

At the beginning of $\mathcal{I}_i$, at time τ_i, A is aware of two requests, say σ_h and σ_k. Without loss of generality, suppose that A is located in P (where it has just served a request): then, one new request will be released in P at time $\tau_i + \Delta$. Now, A must select the next request it is going to serve. If A decides to serve the request in P, it has to wait up to time $\tau_i + \Delta$ at least. If A decides to serve the request in Q, it has to travel to point Q, thus the request will be served not earlier than time $\tau_i + d$. This implies that, in any cases, $\tau_{i+1} - \tau_i \geq \min\{\Delta,\ d\} = d$. Likewise, $\tau_{i+2} - \tau_{i+1} \geq d$.

We have just showed that $\|\mathcal{I}_i\| \geq 2d$. Since $2d > \Delta$, at least one request among σ_h and σ_k is served with a minimum delay of $2d - \Delta$. Notice that both the requests are not necessarily served inside $\mathcal{I}_i$; but the cost $2d - \Delta$ is incurred by A totally inside $\mathcal{I}_i$. If one request among σ_h, σ_k is still active at the beginning of $\mathcal{I}_{i+2}$, then it will also contribute to the cost incurred by A during interval $\mathcal{I}_{i+2}$.

Since there are m time intervals, we have that:

$$\mathsf{A}(\sigma) \geq m(2d - \Delta).$$

The off-line adversary B has the following behavior. Without loss of generality, suppose that A serves σ_1 before σ_2.

(1) At time Δ, B serves σ_2 in Q, at no cost.
(2) At time $\Delta + d < 2\Delta$, B serves σ_1 in P, paying d. All starting requests have been served.
(3) Recall that A serves its first request in P, at time $\tau_1 \geq \Delta$. B waits in P until time $\tau_1 + \Delta\ (\geq 2\Delta)$; then B begins to follow A with a delay of Δ, i.e. $p_{\mathsf{B}}(t) = p_{\mathsf{A}}(t - \Delta)$, for all $t \geq \tau_1 + \Delta$. While traveling, B serves all the requests it encounters. Notice that these requests are served at no cost, since they are released in a point (P or Q) with a delay of Δ with respect to the time when A was in the same point.

Hence we have that $\mathsf{B}(\sigma) = d$, and the competitive ratio $\mathsf{A}(\sigma)/\mathsf{B}(\sigma)$ can be made arbitrarily large with a suitable choice of m. □

The above negative results for the net latency force us to switch back again to the easier objective function of latency. For the interval metric space, we show that algorithm OptimizeEarlierRequestsOnly, defined in Section 3.2, has a competitive ratio which tends to 1 as Δ increases.

Theorem 11. *If f is the latency,* $\mathsf{OptimizeEarlierRequestsOnly}_f$ *with time lookahead $\Delta = \delta D$ is a $(1 + 2/\delta)$-competitive algorithm for* L-OL-TRP *on an interval of length D.*

Proof. Let $\sigma = \sigma_1 \ldots \sigma_n$ be the input instance, with $\sigma_i = (x_i,\ t_i)$; let τ_i and τ_i^* denote the time when request σ_i is served by OERO and OPT respectively. We partition the set of input requests into three subsets, and denote by A_1, A_2 and A_3 their indices:

- A_1 is the set of indices of the requests released before time Δ, and served when OERO is in the initial mode;
- A_2 is the set of indices of the requests released before time Δ, and served when OERO is in the sweeping mode; we denote with τ_i^{**} the time when these requests would have be served if OERO remained in the initial mode;
- A_3 is the set of indices of the requests released after time Δ.

We have that $\mathsf{OERO}(\sigma) = \sum_{i\in A_1} \tau_i + \sum_{i\in A_2} \tau_i + \sum_{i\in A_3} \tau_i$; and $\mathsf{OPT}(\sigma) = \sum_{i\in A_1} \tau_i^* + \sum_{i\in A_2} \tau_i^* + \sum_{i\in A_3} \tau_i^*$. We first note that

$$\sum_{i\in A_1} \tau_i + \sum_{i\in A_2} \tau_i^{**} \leq \sum_{i\in A_1} \tau_i^* + \sum_{i\in A_2} \tau_i^*,$$

since the first term is the *optimal* cost of the solution which serves *only* the requests released before time Δ.

Consider now any request σ_i with $i \in A_2$. Since the server switched to the sweeping mode, σ_i was not served before time Δ: thus $\tau_i^{**} \geq \Delta = \delta D$. But we have that $\tau_i \leq \tau_i^{**} + 2D \leq (1 + 2/\delta)\tau_i^{**}$, since (i) the server switched to the sweeping

mode before time τ_i^{**} and (ii) after switching, the server served all the requests in A_2 within the first complete sweep of the segment. Thus

$$\sum_{i\in A_1}\tau_i+\sum_{i\in A_2}\tau_i\le\sum_{i\in A_1}\tau_i+\sum_{i\in A_2}\left(\left(1+\frac{2}{\delta}\right)\tau_i^{**}\right)\le\left(1+\frac{2}{\delta}\right)\left(\sum_{i\in A_1}\tau_i^{*}+\sum_{i\in A_2}\tau_i^{*}\right).$$

For the requests in A_3, we have that $\tau_i^*\ge t_i\ge\Delta=\delta D$, and $\tau_i\le t_i+2D$: hence $\tau_i\le(1+2/\delta)\tau_i^*$. We conclude that

$$\begin{aligned}\mathsf{OERO}(\sigma)&=\sum_{i\in A_1}\tau_i+\sum_{i\in A_2}\tau_i+\sum_{i\in A_3}\tau_i\\&\le\left(1+\frac{2}{\delta}\right)\left(\sum_{i\in A_1}\tau_i^{*}+\sum_{i\in A_2}\tau_i^{*}\right)+\sum_{i\in A_3}\left(\left(1+\frac{2}{\delta}\right)\tau_i^{*}\right)\\&=\left(1+\frac{2}{\delta}\right)\mathsf{OPT}(\sigma).\quad\square\end{aligned}$$

5. Concluding remarks: Lookahead and the virtues of laziness

In the earlier sections, we presented several on-line vehicle routing problems, and we studied the impact that lookahead can have on their competitive ratio. This impact varies considerably across problems and metric spaces. As we have seen in Section 3, in general metric spaces, lookahead does not provide any extra power to on-line algorithms for the H-OL-TSP, but it helps algorithms for the N-OL-TSP, allowing us to improve the upper bound from $1+\sqrt{2}$ to 2 (when a sufficient amount of lookahead is given). If the metric space is the line segment, lookahead becomes much more useful, as it allows the competitive ratio to tend to 1 as the amount of lookahead increases. This holds for the homing and the nomadic OL-TSP, as well as for the OL-TRP. If we consider the net latency as the objective function we immediately run into negative results: there is no on-line competitive algorithm in any space containing a bidimensional ball, and even in a unidimensional space there is no hope of obtaining a competitive on-line algorithm if the amount of lookahead is not large enough.

Another direction that has been explored in other works [9,10,17,26,30], and that we overviewed in Section 2, is whether and when it is advantageous for an algorithm to wait idle even when there are outstanding requests. Even though it may sound unusual that an algorithm should decide to wait instead of serving pending requests, as discussed in Section 2 moving the server too early could damage the quality of the overall service. In particular, we can observe from Tables 1 and 2 that for the same problems where lookahead proves useful (i.e. for the N-OL-TSP in general metric spaces, for the H-OL-TSP, the N-OL-TSP and the L-OL-TRP on the line), non-zealous algorithms behave better than the best zealous algorithms known. Conversely, for the H-OL-TSP, where lookahead is useless, the optimal algorithm is a zealous one. In this concluding section, we would like to make some more precise remarks on the relationship between lookahead and the non-zealousness (or laziness) of algorithms.

To this end, consider any real-time problem with the following property: if we take a solution to an instance and delay it by Δ units of time, then the cost of the new solution on the same instance increases by at most Δ. For example, the nomadic and the homing OL-TSP have this property. For such "makespan-type" problems there is an intuitive connection between lookahead and waiting, because an algorithm A could always wait up to time Δ, simulate an algorithm B with lookahead Δ on the same instance and apply B's solution with delay Δ. In particular, if the competitive ratio of B with lookahead $\delta\cdot\mathsf{OPT}$ is $\rho(\delta)$, and A was somehow able to take $\Delta=\delta\cdot\mathsf{OPT}$, then A would be $\delta+\rho(\delta)$ competitive. Thus, the effectiveness of lookahead implies the effectiveness of waiting. In our framework this connection is not formal, of course, because it is not clear how A can guess the right Δ (which is instance-dependent). But in retrospect, an important component of many real-time algorithms is precisely the on-line estimation of the optimal cost. Furthermore, we remark that, while we decided to relate time lookahead with the diameter of the metric space, an alternative way to give a meaning to lookahead is to compare it with the optimal cost of the input instance.

This is, essentially, the approach adopted by Jaillet and Wagner [23]: they compare time lookahead Δ with the length of the optimal tour L_{TSP}. For the H-OL-TSP they prove that, if $\Delta=\alpha L_{TSP}$, then there exists a $\left(2-\frac{\alpha}{1+\alpha}\right)$-competitive algorithm for general metric spaces, thus improving on the 2-competitive result of [9]. Notice that, as a consequence of Theorem 4, this result crucially depends on the fact that lookahead is not fixed a priori, but is a function of the input instance. For the latency objective function, Jaillet and Wagner compare Δ with both L_{TSP} and t_n, where t_n is the time when the last request is released: they extend the best known algorithm of [27], using lookahead to improve its competitive ratio. The idea of comparing Δ with characteristic quantities of the instance, such as its optimal cost, makes it possible to usefully apply lookahead in any metric space, and has a theoretical interest because, by regulating lookahead parameters, one can vary the amount of "on-lineness" of the model. On the other hand, it is hardly a practical approach, as it requires input instances to conform to some rules (for example, to disclose requests with a lookahead that is proportional to the length of the optimal tour) which do not seem to be easily justifiable in a real-world application.

We finally remark that, while theoretically the influence of lookahead varies significantly among problems and metric spaces – and in several cases lookahead proves scarcely useful or completely useless – practically algorithms can make a

good use of the information provided by lookahead. In [4] we present preliminary experimental results, where we show that, in every problem and metric space we considered, even small amounts of lookahead considerably improve the average competitive ratios of the simple algorithms analyzed.

Acknowledgement

Third author's work was partially supported by the Future and Emerging Technologies Unit of EC (IST priority — 6th FP), under contract no. FP6-021235-2 (project ARRIVAL).

References

[1] F. Afrati, S. Cosmadakis, C.H. Papadimitriou, G. Papageorgiou, N. Papakostantinou, The complexity of the travelling repairman problem, Theoretical Informatics and Applications 20 (1) (1986) 79–87.
[2] S. Albers, On the influence of lookahead in competitive paging algorithms, Algorithmica 18 (3) (1997) 283–305.
[3] S. Albers, A competitive analysis of the list update problem with lookahead, Theoretical Computer Science 197 (1–2) (1998) 95–109.
[4] L. Allulli, G. Ausiello, L. Laura, On the power of lookahead in on-line vehicle routing problems, Technical Report TR-02-05, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 2005.
[5] N. Ascheuer, S.O. Krumke, J. Rambau, Online dial-a-ride problems: Minimizing the completion time, in: H. Reichel, S. Tison (Eds.), Proc. 17th Symp. on Theoretical Aspects of Computer Science, in: Lecture Notes in Computer Science, vol. 1770, Springer, 2000, pp. 639–650.
[6] G. Ausiello, L. Allulli, V. Bonifaci, L. Laura, On-line algorithms, real time, the virtue of laziness, and the power of clairvoyance, in: J. Cai, S.B. Cooper, A. Li (Eds.), Proc. 3rd Int. Conf. on Theory and Applications of Models of Computation, in: Lecture Notes in Computer Science, vol. 3959, Springer, 2006, pp. 1–20.
[7] G. Ausiello, V. Bonifaci, L. Laura, The on-line asymmetric traveling salesman problem, Journal of Discrete Algorithms 6 (2) (2008) 290–298.
[8] G. Ausiello, M. Demange, L. Laura, V. Paschos, Algorithms for the on-line quota traveling salesman problem, Information Processing Letters 92 (2) (2004) 89–94.
[9] G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, M. Talamo, Algorithms for the on-line travelling salesman, Algorithmica 29 (4) (2001) 560–581.
[10] M. Blom, S.O. Krumke, W.E. de Paepe, L. Stougie, The online-TSP against fair adversaries, INFORMS Journal on Computing 13 (2001) 138–148.
[11] V. Bonifaci, An adversarial queueing model for online server routing, Theoretical Computer Science 381 (1–3) (2007) 280–287.
[12] V. Bonifaci, Models and algorithms for online server routing, Ph.D. Thesis, Technical University Eindhoven, The Netherlands, 2007. Available online at: http://www.dis.uniroma1.it/~bonifaci/papers/phdthesis-tue.pdf.
[13] V. Bonifaci, L. Stougie, Online k-server routing problems, Theory of Computing Systems, in press (doi:10.1007/s00224-008-9103-4).
[14] A. Borodin, R. El-Yaniv, Online Computation and Competitive Analysis, Cambridge University Press, 1998.
[15] A. Borodin, J.M. Kleinberg, P. Raghavan, M. Sudan, D.P. Williamson, Adversarial queuing theory, Journal of the ACM 48 (1) (2001) 13–38.
[16] W. de Paepe, Complexity results and competitive analysis for vehicle routing problems, Ph.D. Thesis, Technical University of Eindhoven, 2002.
[17] E. Feuerstein, L. Stougie, On-line single-server dial-a-ride problems, Theoretical Computer Science 268 (2001) 91–105.
[18] A. Fiat, G.J. Woeginger (Eds.), Online Algorithms: The State of the Art, Springer, 1998.
[19] N. Garg, A 3-approximation for the minimum tree spanning k vertices, in: Proc. 37th Symp. Foundations of Computer Science, 1996, pp. 302–309.
[20] M. Goemans, J. Kleinberg, An improved approximation ratio for the minimum latency problem, Mathematical Programming 82 (1) (1998) 111–124.
[21] R.L. Graham, Bounds for certain multiprocessing anomalies, Bell System Technical Journal 45 (1966) 1563–1581.
[22] D. Hauptmeier, S.O. Krumke, J. Rambau, The online dial-a-ride problem under reasonable load, in: Proc. 4th Italian Conference on Algorithms and Complexity, in: Lecture Notes in Computer Science, Springer, 2000, pp. 125–136.
[23] P. Jaillet, M.R. Wagner, Online routing problems: Value of advanced information as improved competitive ratios, Transportation Science 40 (2) (2006) 200–210.
[24] M. Jünger, G. Reinelt, G. Rinaldi, The traveling salesman problem, in: M.O. Ball, T. Magnanti, C.L. Monma, G. Nemhauser (Eds.), Network Models, Handbook on Operations Research and Management Science, vol. 7, Elsevier, North Holland, 1995, pp. 225–230.
[25] E. Koutsoupias, C.H. Papadimitriou, Beyond competitive analysis, SIAM Journal on Computing 30 (1) (2000) 300–317.
[26] S.O. Krumke, Online optimization: Competitive analysis and beyond. Habilitation Thesis, Technical University of Berlin, 2001.
[27] S.O. Krumke, W.E. de Paepe, D. Poensgen, L. Stougie, News from the online traveling repairman, in: Proc. 28th International Colloquium on Automata, Languages, and Programming, 2001, pp. 487–499.
[28] S.O. Krumke, L. Laura, M. Lipmann, A. Marchetti-Spaccamela, W.E. de Paepe, D. Poensgen, L. Stougie, Non-abusiveness helps: An $o(1)$-competitive algorithm for minimizing the maximum flow time in the online traveling salesman problem, in: Proc. 5th Int. Workshop on Approximation Algorithms for Combinatorial Optimization, 2002, pp. 200–214.
[29] L. Laura, Risoluzione on-line di problemi dial-a-ride, Master's Thesis, University of Rome "La Sapienza", 1999.
[30] M. Lipmann, On-line routing, Ph.D. Thesis, Technical University of Eindhoven, 2003.
[31] R. Sitters, The minimum latency problem is NP-hard for weighted trees, in: Proc. 9th Integer Programming and Combinatorial Optimization Conference, 2002, pp. 230–239.
[32] D. Sleator, R.E. Tarjan, Amortized efficiency of list update and paging rules, Communications of the ACM 28 (2) (1985) 202–208.